Artifact Detection and Evaluation in JPEG compressed images

Călin Borlovan

July 20, 2015

One-year programme Master's Software Engineering

Supervisor: dr. Magiel Bruntink
Host organization: Voormedia B.V.

UNIVERSITY OF AMSTERDAM

voormedia

Abstract

JPEG is a widely used lossy compression method in the field of image processing. One of the main disadvantages of its block-based DCT compression is the introduction of visible artifacts in the compressed image due to a quantization of the frequency coefficients. Between the most common types of artifacts are the blocking, ringing and mosquito noise artifacts. In this paper I will present the results of a research project conducted at Voormedia B.V. in order to find out ways of detecting the mentioned types of artifacts in the spatial domain of a compressed image, quantifying their appearance and relating the resulted measures to the levels of acceptable quality.

Contents

1 Introduction

In the world of image processing one of the important research topics is the one of image compression. For example, image storage cost and transmission speed are two relevant aspects that are directly correlated to the size of an image. In order to efficiently carry out the two mentioned processes, it is usually desired that the image data is stored in an optimal way. In this sense multiple ways of encoding an original image in order to reduce its size have been developed, each taking a different approach.

However, the algorithms can be classified in two main categories: lossless compression algorithms and lossy compression algorithms. When compressing an image through a lossless approach the original image can be totally reconstructed from the compressed one. In contrast, a lossy compression method allows better compression rates by discarding some of the information in the image and thus making the recovery of the original image out of a compressed one impossible. The reason for reducing the amount of information in an image comes from the idea that the human visual system is usually incapable of perceiving all the detail in an image. A good example is the fact that the human eye can perceive a smaller spectrum of colors than the one that can be represented in a digital image.

In the context of lossy compression, one of the widely used standards in today's digital world is JPEG. One of the tools developed by Voormedia B.V. is named TinyJPG and serves the scope of reducing the size of an image as much as possible, while keeping its perceived quality at acceptable levels. The highest goal in the context of this tool would be to compress the image as far as the differences perceived by the human eye from the original image when visualized at its real size are considered acceptable. During this study we have been looking at ways of optimizing the compression process by analyzing the already compressed image.

1.1 Motivation and Problem Statement

TinyJPG gives highly satisfying results when compressing images, but we consider that its performance can be even more improved. Some of the images can be compressed more, while others less depending on their characteristics and one of the main driving factors in establishing the perceived quality of the compressed image is the level of distortion caused by artifacts. The process of TinyJPG takes an iterative approach by compressing several times the image until the desired level of quality and size is reached. Thus we consider that by detecting and quantifying the artifacts in a compressed image, the quality settings could be adjusted to get better results in a further iteration.

During this project we will first try to identify what are the current methods used in detecting the most common types of artifacts, blocking, ringing and mosquito noise. We will further on investigate whether any of these solutions fit the context of TinyJPG and try to replicate and adjust them in this sense.

Once methods for detecting artifacts have been defined, the next step would be to identify a way of quantifying the affected regions. Finally we will investigate whether measuring the artifacts of a compressed image can actually be a good indicator on how far the compression level can be pushed on an image.

1.2 Research Questions

The research questions that we will try to answer during this project are:

- Can we appropriately measure how much a JPEG compressed image is affected by artifacts?
- Can the quantification of artifacts (their amount, visibility estimation) be a good indicator on the acceptability of a compressed image?
- Can this information be used to improve on existing techniques that decide on the quality of a compressed image by using similarity metrics?

1.3 Outline

The current chapter gives and introduction to the context and purpose of this research project. In the next section background information that is necessary for understanding the main concepts in JPEG compression is presented and the third section will outline the design of the research that has been carried out. In the fourth chapter the mathematical foundation and code snippets of the implemented solution will be outlined. The fifth chapter describes an experiment that has been carried out, while the sixth chapter discusses evaluation and validation results. Finally, the paper will end up with a short analysis and conclusion of the entire research.

2 Background

2.1 JPEG Standard

JPEG is a standard of lossy compression in computer vision that stands for Joint Photographic Experts Group, the committee that created it. A lossy compression represents a method of representing data by discarding some of its information to reduce the amount that needs to be stored. In contrast to a lossless compression method, the lossy ones are often able to reduce the data much more. On the other side, the discarded information is impossible to be restored and thus the quality can decrease significantly based on the compression level.

The JPEG encoding takes place in several steps, but before usually some preprocessing actions can be carried out to prepare the image for a more efficient compression:

- **Color space transformation**

  The image is converted during this step from the RGB space to the Y'CbCr space. In this space the luminance of a pixel is represented by the Y' component, while the Cb and Cr represent the chrominance split into blue and red components. This conversion is carried out as the Y'CrCb space allows a greater compression that keeps the effect on the perceived image quality less significant. The human visual system has been proved to be more sensitive to changes in brightness than color and by collecting all the brightness information on one channel there are better premises for an efficient compression.

- **Downsampling**

  As the human visual system can't see the finer details in the hue and color saturation of an image than it usually exists, another transformation that can improve the compression is to reduce the spatial resolution of the chroma components. This step is known as downsampling and it can be done at different ratios, from which the most common ones for JPEG images are 4:4:4 (no downsampling), 4:2:2 (reduction by a factor of 2 in horizontal direction) or 4:2:0 (reduction by a factor of 2 in both horizontal and vertical directions)

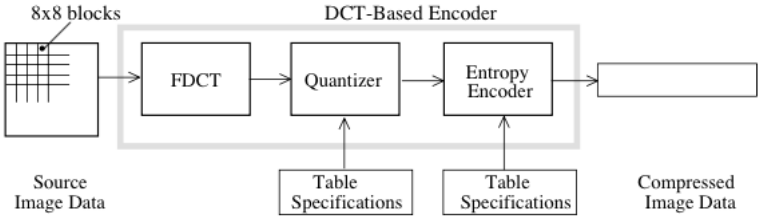


Figure 2: DCT-Based Encoder Processing Steps [1]

Once all the preprocessing steps have been completed, the image data is split for each channel of the image into blocks of 8x8 that will represent the input of the actual encoder. Most of the time it can of course be the case that some of the marginal blocks would be incomplete and thus some data must be filled. A common approach can be in this case replicating the edge pixels.

The stages of the JPEG encoder, as they can can be visualized in Figure 2, are:

- **8x8 Forward DCT**

  Each of the resulting blocks and for each component is converted to the frequency domain by using the Forward Discrete Cosine Transformation (FDCT). Firstly, this transformation requires a preparation step that for an 8-bit image (each entry in [0,255]) consists of shifting all the values from a positive range to one centered around zero. The value 128 is subtracted from each entry

and the new formed range will be [-128,127] [1]. Next, the two dimensional DCT is computed by the following equation:

$$F_{u,v} = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f_{x,y}\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right] \tag{1}$$

where:

- u and v represent the horizontal and vertical spatial frequencies
- C(u), C(v) $=\frac{1}{\sqrt{2}}$ for u,v=0;
  C(u), C(v) = 1 otherwise.
- $f_{x,y}$ the pixel value at coordinate (x,y)

235.6	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3
-22.6	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2
-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1
-7.1	-1.9	0.2	1.5	0.9	-0.1	0.0	0.3
-0.6	-0.8	1.5	1.6	-0.1	-0.7	0.6	1.3
1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0
-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8
-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4

Figure 3: DCT coefficients example [1]

The resulting matrix of this transformation contains the DCT coefficients that represent the amount of 2D spatial frequencies similar to Figure 3. The coefficient with zero frequency is called the DC coefficient, while the other 63 represent the AC coefficients [1].

- **Quantization**

  The purpose of this step is to reduce the precision of the DCT coefficients so to be able to represent the image at the desired quality. The quantizer thus takes as an input the 64 coefficients resulting from the previous stage and by using the 64-element Quantization Table that is passed as an input parameter to the encoder, it uniformly quantizes them all. This step represents the main source of information loss in the JPEG encoding [1].

  The quantization formula can be described as follows:

$$F^{Q}(u,v) = \text{round}\left(\frac{F(u,v)}{Q(u,v)}\right) \tag{2}$$

  The results of the quantization step can be observed in Figure 4 and note the amount of zeros in the resulting matrix.

- **Entropy coding**

  Before the actual entropy encoding step starts, two other processes take place.

  First the DC coefficient is calculated as well, but under a different form. The DC component represents the average luminance/chrominance in the current block and contains an important

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

(a)

15	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b)

Figure 4: (a) Quantization Table example; (b) Quantized coefficients [1]

amount of the image's information. As these coefficients are usually strongly correlated between adjacent blocks (more or less the same), the DC coefficient is encoded as a difference from the DC coefficient of the previously encoded block [1].

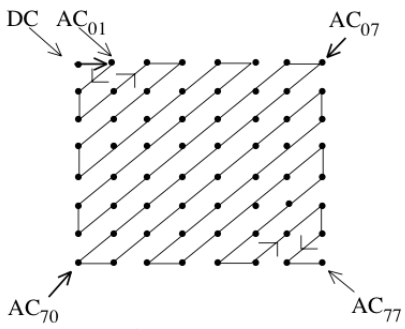


Figure 5: Zig-zag sequence of quantized coefficients [1]

Furthermore, all the quantized DCT coefficients are aligned in a "zig-zag" manner in order to facilitate the entropy encoding by placing the low-frequency before the high-frequency ones, similar to Figure 5.

The entropy coding itself is a lossless approach that puts the coefficients in a more compact way. The main two methods proposed by JPEG for carrying out this step are the Huffman coding and the arithmetic coding. However, the arithmetic encoding is not used that much anymore as it has been covered by patents. Additionally, it has proved to be slower than the Huffman encoding.

2.2 Compression Artifacts

One of the main disadvantages of a lossy compression method is of course the fact that the data once discarded can introduce a loss of quality. As a consequence, at high compression levels, the loss of too much information can facilitate the appearance of distortions known as compression artifacts. Several types of artifacts can be identified, having different causes, but out of these categories the most frequent and prominent ones are the blocking, ringing and mosquito noise artifacts.

2.2.1 Blocking Artifacts

Blocking artifacts are common distortions that appear in a JPEG compressed image due to the block-based processing of the encoding algorithm that is not capable to take into consideration correlations between neighbouring blocks. They manifest themselves in the reconstructed image as discontinuities at the border between two neighbouring blocks and are especially visible in smooth areas of an image. In areas with a lot of detail the presence of these distortions is often masked and thus less annoying for the human eye.

The occurrence of such artifacts is a direct result of a high compression rate that discards too many frequencies in the blocks of an image. In areas with a low gradient, the direct consequence is that the gradient at the block's border will increase with the increase of the compression level.

(a) (b)

Figure 6: (a) Original lena.jpg image; (b) Compressed lena.jpg image at quality 10 and with visible blocking artifacts

An example of how blocking artifacts appear in a JPEG image can be observed in Figure 6.

2.2.2 Ringing Artifacts and Mosquito Noise

In areas of an image that present a sharp transition from low to high frequency, two more types of distortions can become visible at high compression rates: ringing artifacts and mosquito noise. As the two types or artifacts are spatially connected, for example appearing around edges of an object placed in a smooth area, they are often referred only as ringing. However their appearance is different and the causes differ as well.

Ringing artifacts have the appearance of "ghosts" around edges and are mathematically related to the Gibbs phenomenon. They appear due to bandlimited frequencies in the image once the quantization of the encoder cuts the highest ones. In the lack of high frequencies, the step transitions around sharp edges start to present bumps or more formally, the step response of the quantization process presents oscillations.

Mosquito noise artifacts appear in the same regions as the ringing, but while ringing can be present also in other signal processing applications, the mosquito noise is typical only to JPEG and is also a direct consequence of the block-based processing. Their appearance is the one of dots that follow the edges in an image and in an 8x8 JPEG encoding they can appear as far as 7 pixels away from an edge. In other words, they can appear anywhere in a block that contains edges. Same as for blocking artifacts, their annoyance is much higher in smooth areas that contain edges, rather than in high frequency areas where their effect is masked by the amount of detail in the image.

(a) (b)

Figure 7: (a) Original image; (b) Compressed image at quality 20 and with visible ringing and mosquito noise artifacts

An example image that at high compression levels introduces ringing and mosquito noise artifacts can be visualized in Figure 7.

2.3 Related work on artifact detection/reduction

The reduction of artifacts in JPEG compressed images has received great attention in the research world and thus several approaches have been developed. Reducing artifacts on a already compressed image first requires an identification of the affected areas. In order to ensure that no artifacted regions are missed in the detection and at the same time that no textural elements of the image are labeled as artifacts, a very good reasoning needs to be applied. As there hasn't yet been a clear definition of how an artifact manifests in the image data, the best one can do is to reduce as much as possible the amount of false positives and the misses. Due to their different nature, the artifacts need different approaches to be detected. Ringing and mosquito noise are however often referred just as ringing due their overlapping root causes and thus the same detection systems are used. Furthermore, the identified reduction schemes differ a lot in their approach. We will briefly present a few of the related papers on this topic and mention why some of them fit or don't fit the purpose of the current research project.

Minamo et al [3] introduced a new metric for detecting blockiness named Mean Square Difference of Slope (MSDS). The same metric has been afterwards used also by Lakhani et al [4] and Triantafyllidis et al [5]. The approaches try to first locate blocked regions through the MSDS and afterwards reduce blockiness by minimizing the value of the MSDS for those regions. The minimization is carried out through changes in the DCT coefficients of the encoder. Due to the reduction in the frequency domain which is not possible in the current context (black-box encoder), only the artifact detection can be a viable possibility for our purposes.

Eerenberg et al [2] proposed two detection systems for locating ringing artifacts in DCT-based compressed images and videos. One of the systems is implemented in the spatial domain, while the other one in the frequency domain. Again, only the detection in the spatial domain is of interest for us. The main idea of this block-based approach is to identify artifacts by performing a spatial reasoning on the surrounding area of an image block.

Liu et al [6, 7] proposed another method for detection in the spatial domain of ringing artifacts. The method first performs an edge detection on the image and afterwards a ringing detection around the detected edges. A ringing visibility estimation is added to the detection mechanism to asses the annoyance of the artifacts. Concepts of both papers can be a good fit for the current context.

Once the affected regions are identified most of the methods of reducing these distortions propose some sort of post-processing techniques such as filtering. A review of some of the methods is presented by Shen et al [12]. The operations mostly consist of smoothening (e.g. the method of Kim et al [16]), but

the results often introduce bluriness to mask the artifacts. Moreover, the post-processing of an image might reduce the visual annoyance of the artifacts, but at the same time it can increase the size of an image. In the context of TinyJPG where reducing the size as much as possible and keeping the similarity to the original image very close, post-processing might not be a viable solution. One such method is proposed by Popovici et al [9] which even tries to do a controlled filtering on the affected regions, but this still doesn't avoid all its consequences.

Luo et al [10] and Fang [13] introduced detection methods for blocking artifacts in the DCT domain, followed again by a filtering to reduce the contaminated areas. The approaches in both of the phases are not a good fit in our context.

Another category of techniques prefer to rather use the original image to avoid the introduction of artifacts. These methods either try to preprocess the image through some enhancement methods, or try to predict where artifacts might be prone to appear. For example, once the sensitive areas have been identified, the encoding process can be tuned accordingly by adjusting the quantization tables or the DCT matrix. The approach of analyzing the input image of the compression is however the scope of another research project conducted here at Voormedia B.V. and thus such methods won't be further on explored in this paper.

One more different category of papers that focus on artifact detection and reduction propose methods that include machine learning concepts. One such example is the approach introduced by Zhang et al [11] which makes use of an adaptive neural network-based algorithm to reduce blockiness in JPEG compressed images, or the method of Chang et al [15]. Even though the results of such an approach can sometimes be promising, due to its complexity and lack of specialization in the field we have decided to consider machine learning approaches just as possible future work.

3 Research Design

Answering the proposed research questions implies the use of different research methods that would guide the process. We will present in the followings the main research processes that will be conducted for the specific questions.

- Can we appropriately measure how much a JPEG compressed image is affected by artifacts?

  Answering this question implies a good understanding of what JPEG artifacts represent and then an identification of the currently existing methods of detecting and measuring artifacts. Thus the conducted research processes will be:

  - **Artifact reproduction**

    A first logical step is to understand what types of artifacts exist, how does their appearance in an image look like, to what extent are they annoying for the human perception, where are the areas in the image where they are more likely to appear or at least where they are more visible. To cover these aspects we will experiment with mozjpeg, a common JPEG encoder, by compressing images from different categories at different qualities. The monitored variables will be the quality levels at which the image starts being unacceptable, the regions where blocking, ringing and mosquito noise artifacts appear.

  - **(Partial) Replication**

    In our attempt to answer the first research question, we will create based on the solutions identified during the literature survey that fit best the context of the research project two prototypes, one for an algorithm detecting blocking artifacts and another one for detecting ringing and mosquito noise. The chosen approaches will be presented in Section 4.

- Can the quantification of artifacts (percentage, annoyance score) be a good indicator on the acceptability of a compressed image?

  The main challenge for answering this research question will be to actually understand what users consider to be an acceptable compressed image. Afterwards we will try to identify relations between the acceptability of an image and the compression artifacts. The conducted processes will be:

  - **Psychovisual human experiment**

    The experiment will be conducted in order to understand for a predefined set of images what are the acceptable quality levels. The results will be used to set up the sensitivity of the detection algorithms and also for creating a model to estimate the acceptability of an image. The details of this experiment will be presented in Section 5

  - **Hypothesis validation**

    The estimation model for the perceived acceptability of an image will be validated through a second experiment that will have a similar setup as the previously mentioned one, but it will be executed on a different set of images. The validation process will be presented in Section 6.

- Can this information be used to improve on existing techniques that decide on the quality of a compressed image by using similarity metrics?

  Answering the last proposed question will imply an analysis of all the research results and some more artifact measurements on different types of images. The details will be presented in Section 6.

4 Artifact detection and evaluation

When looking at a JPEG compressed image, blocking and ringing artifacts can be observed to appear in different parts of an image and have different appearance. This aspect gives a strong clue on the fact that the two types of artifacts are fundamentally different and thus require different approaches at least in detecting them and perhaps even in evaluating their impact on the perceived quality.

We will thus outline two methods for detecting artifacts in a compressed image that partially replicate solutions from other papers, one for blocking artifacts based on the approach of Minamo et al [3] and one for ringing/mosquito noise artifacts based on the approach of Eerenberg et al [2]. The two approaches take concepts of the mentioned papers, adapt them to our current needs and also introduce some new ideas. The programming language that we have used for prototyping the proposed solution is Python, due to its very good support for image processing through libraries such as NumPy and OpenCV. The JPEG compression encoder is the Mozilla JPEG encoder, also simply known as mozjpeg.

4.1 Blocking artifacts

Blocking artifacts, as presented in the previous chapter, are a direct consequence of the block-based approach used in the JPEG encoder. It would make sense in this context that a block-based detection algorithm can be applicable for highlighting blocking artifacts. The method introduced by Minamo et al [3] consisted of two parts, one of detecting artifacts and a second one of reduction in the frequency domain. As the encoder is a black-box in our context, the reduction as presented by Minamo et al [3] wouldn't be possible. We have thus focused only on the detection method.

As this type of artifacts manifests itself as discontinuities in terms of gradient that appear between neighbouring blocks, the proposed detection approach was to first split the image into blocks of 8x8 pixels similar to the block size used in encoding and afterwards for each region its blockiness with regards to its four neighbouring regions will be evaluated (Figure 8). The blockiness was evaluated by using the metric Mean Square Difference of Slope.

We considered the detection mechanism of Minamo et al [3] as a good fit also for our scope due to its simplicity and application in the spatial domain and took its approach, with a few changes and additions. First of all, the Mean Square Difference of Slope has been replaced in our approach by a Mean Absolute Difference of Slope as this aspect made the threshold reasoning much simpler and also gave an equal importance to all the pixel differences. Secondly, we will introduce an additional set of constraints that we consider necessary for an efficient detection that reduces the amount of false positive detection.

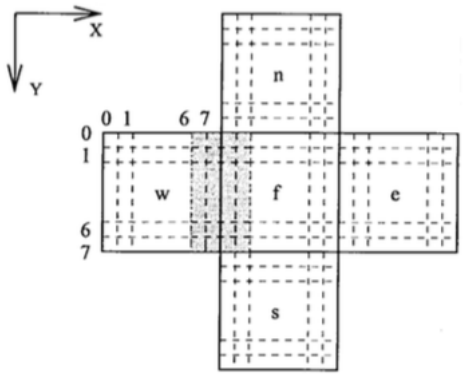


Figure 8: Adjacent Blocks in Blocking Artifact Detection [4]

4.1.1 Mean Absolute Difference of Slope (MADS)

The Mean Absolute Difference of Slope, as mentioned before, is taking the concept of the Mean Square Difference of Slope [3]. We will thus present the idea behind the MSDS and afterwards mention the small difference in the MADS.

The hypothesis of the MSDS is that when doing a JPEG block-based compression, the gradient slope across block boundaries is increased and thus the intensities here as well, becoming visible to the human eye. This is however an unusual aspect as most of the images (especially natural ones) contain smooth areas that don't present such discontinuities [3].

An obvious approach in this context of detecting blocked edges between two neighbouring blocks is to compute the difference of gradient on these edges and evaluate it to determine whether it can labeled as artifacted or not. The concept behind the MSDS is thus to calculate the difference between the slope across block boundaries and the average of the slopes of the two adjacent blocks close to the boundary.

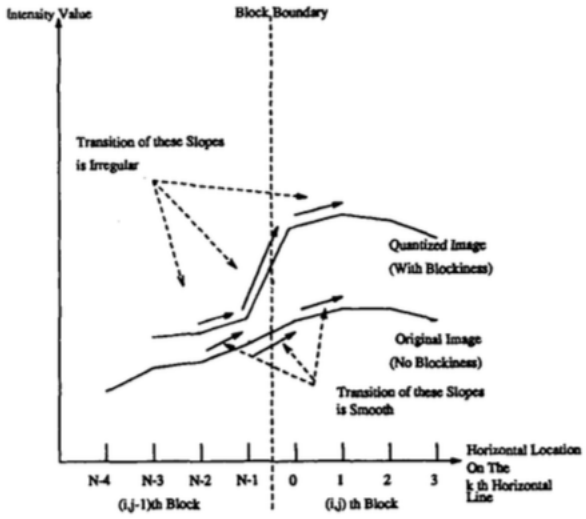


Figure 9: Differences of Slope [3]

The reason why the MSDS is a good measure for checking the blockiness and why it appropriately reflects the discontinuities along block boundaries can be depicted from Figure 9.

The only difference between the MSDS and the MADS consists in its computational formula, where the MADS uses, as the name suggests, an absolute difference instead of a squared one. The rest of the concepts have also been taken from the method of Minamo et al [3]. As a consequence, the equation of the MADS is:

$$\text{MADS}_{i,j} = \frac{1}{N} \sum_{k=0}^{N-1} \left| d_{i,j}(k) - d'_{i,j}(k) \right| \tag{3}$$

where for two neighboured blocks in horizontal direction:

- $d_{i,j}(k)$ - the boundary slope with the formula:

$$d_{i,j}(k) = x_{i,j}(k,0) - x_{i,j-1}(k,N-1) \tag{4}$$

- $d'_{i,j}(k)$ - the average of the slopes next to the boundary

$$d'_{i,j} = [d''_{i,j-1}(k,N-1) + d''_{i,j}(k,1)]/2 \tag{5}$$

$$d''_{i,j}(k,l) = x_{i,j}(k,l) - x_{i,j}(k,l-1) \tag{6}$$

where:

- x - the pixel at the specified location
- N - the block size

The Python code that we have developed to compute the MADS between two neighbouring blocks on a horizontal direction is:

```
total_blockiness = 0

for x in range(0, size):
    boundary_slope = np.abs(second_block[x][0] - first_block[x][size-1])
    first_slope = np.abs(first_block[x][size-1] - first_block[x][size-2])
    second_slope = np.abs(second_block[x][1] - second_block[x][0])

    current_blockiness = boundary_slope - np.float_(first_slope +
        second_slope)/2

    total_blockiness += current_blockiness

total_blockiness = np.float_(total_blockiness)/np.float_(size)
```

The above code loops over all the lines of the two blocks and computes a line blockiness which afterwards is added to the total blockiness. The line blockiness is represented by the difference between the boundary slope and the average of the first block's slope and second block's slope close to the boundary. On one line, the boundary slope is computed through the absolute difference of the last pixel in the first block and the first pixel in the second block. The first block's slope close to the boundary is computed as a difference between the last two pixels on that line, while for the second block the slope close to the boundary is represented by the difference between the first two pixels on the line.

4.1.2 Constraint reasoning

For an accurate detection we have added to the detection method also a set of constraints that would reduce the amount of false positives.

Once the MADS has been computed for two neighbouring blocks, its value needs to be interpreted in order to express whether a boundary edge is artifacted or not. In this sense it can be mentioned that very low values for the MADS can be characteristic to smooth/low intensity areas of an image, while high values can appear in the presence of high intensity regions. On the other side, visible blocking artifacts can be characterized by medium values of the MADS that correspond to slight variations on all three channels of the RGB space. Considering these reasons and by conducting several tests with different images, it made sense to introduce two thresholds, a low and a high one, and label the edges that have a value of the MADS between these two thresholds as being artifacted.

By conducting multiple tests with different images we have observed that boundaries that contain both high frequencies and low frequencies can result in a MADS value that fits the previously set constraint. However, blocking artifacts are not prone to appear on regions such as this, but only in low intensity regions [14]. As a consequence, we have introduced another constraint on the absolute difference of slope stating that the absolute difference of slope of all the N tuples along the edges should not have a larger deviation than 2 from the mean blockiness. This constraint reduces the risk of false positives and still keeps the detection effective by not missing any real artifacts.

$$|\text{MADS}_{i,j} - |d_{i,j}(k) - d'_{i,j}(k)|| \leq 2, k \in \{0, ..., N-1\} \tag{7}$$

Moreover, we have observed during another set of tests that if one of the compared blocks contains high frequencies outside the edge region, false positives are again prone to be detected. We have thus introduced a second constraint stating that the value of the boundary slope and the differences between the boundary pixel of one block and the inner pixels of the second block should allow only slight variations of ± 3.

$$|d_{i,j}(k) - d'''_{i,j}(k,l)| \leq 3, k, l \in \{0, ..., N-1\}, \tag{8}$$

where for checking the boundary pixel of the left first block with the inner pixels of the second block on the horizontal direction the equation for $d'''_{i,j}k, l$ would be:

$$d'''_{i,j}k, l = x_{i,j-1}(k, N-1) - x_{i,j}(k, l) \tag{9}$$

The visualization of the results of the blocking artifact detection algorithm can be observed in Figure 10 where the artifacted edges are marked.

4.1.3 Quantification and Visibility Estimation

Quantifying in an appropriate way the results of the detection algorithm is another aspect of extremely high importance to be able to assess the quality of a compressed image. The goal in estimating the impact of the artifacts on the way the image is perceived by the human eye is to reach some thresholds or combination of thresholds that can be an indicator on how far the compression should go.

The first measure that we have developed is a percentage of artifacted edges with regards to the total amount of block edges in the image. This measure is meant to measure how many visible artifacts are present in an image. However during our tests we have found out that only a percentage of artifacts can't be a good indicator on the quality of an image as it is highly influenced by the structure of an image. For example, an image with a lot of smooth areas is prone to a high percentage of blocking artifacts, while one with less smooth areas will for sure have lower percentages of blocking artifacts. However the images might be equally unacceptable due to the visibility of the artifacts, even though the percentages are so different.

Liu et al [7] introduced the concept of artifact visibility estimation for ringing artifacts, but of course that this concept can be extended also to the blocking artifacts. It makes sense to think not only about the quantity of artifacts, but also on their annoyance to the human eye. We have introduced another measure that for each artifacted edge that has been detected it calculates the average boundary slope and in the end a mean of all the annoyances represents the image annoyance score. The Python function that computes the edge annoyance score is:

```
1 def compute_edge_annoyance(first_block, second_block, direction):
2       if direction == VERTICAL_DIRECTION:
3           return np.average(np.abs(second_block[0:1, 0:BLOCK_COLS]  -
                first_block[BLOCK_ROWS-1:BLOCK_ROWS, 0:BLOCK_COLS]), axis=1)
4
5       if direction == HORIZONTAL_DIRECTION:
6           return np.average(np.abs(second_block[0:BLOCK_ROWS, 0:1] -
                first_block[0:BLOCK_ROWS, BLOCK_COLS-1:BLOCK_COLS]), axis=0)
```

The above code represents a function that takes two blocks of pixels of size BLOCK_ROWS x BLOCK_COLS as parameters and the direction in which the computation of the annoyance score should be made. This means a horizontal direction if the second block is on the right side of the first block in the image and a vertical direction if the second block is below the first one. On a horizontal direction, the last column of the first block and the first one of the second block are used to compute the edge annoyance. The returned score represents the average of the difference between the pairs of pixels from the two columns. In a similar way the annoyance is computed also on a vertical direction.

4.2 Ringing artifacts and Mosquito Noise

While in the case of blocking artifacts the reasoning can be quite straightforward as the only logical condition for detecting them is the presence of small, but abrupt pixel variation in areas of low intensity, for ringing and mosquito noise the entire spatial reasoning can become more complex. The main difficulty in detecting such distortions is differentiating them from actual textural elements of the image or actually defining what they are in terms of pixel variations.

One of the few detection systems in the spatial domain that we have identified during the literature survey was introduced by Eerenberg et al [2]. The main concept behind this method was to first split the image into blocks, compute for each block a representative value called Sum of Absolute Differences (SAD) and afterwards reason on the distortion of each block by considering the entire surrounding region.

(a)

(b)

Figure 10: Blocking artifacts detection (a) up.jpg image compressed at quality 55 and with visible blocking artifacts; (b) Highlighting of the blocking artifacts based on the results of the detection algorithm

The reasoning process implied five different thresholds for the SAD values and a set of boolean equations that represent the constraints for a block to be artifacted. Due to the difficulties imposed on correctly setting the thresholds and missing explanation on the reasoning behind the boolean constraints, we have decided to use some parts of the system as introduced by Eerenberg et al [2] and simplify others.

4.2.1 Sum of Absolute Differences and Detection Kernel

The first step in the approach presented by Eerenberg et al [2] was to split the image into blocks of a configurable size (5x5 pixels chosen in our experiments - different from the size of the JPEG compression blocks so that these don't overlap). Afterwards, for each of our blocks the Sum of Absolute Differences between neighbouring pixels is computed based on the following formula:

$$\text{SAD} = \sum_{y=j}^{M}\sum_{x=i}^{N}\Big|P(x,y) - P(x+1,y)\Big| + \sum_{y=j}^{M}\sum_{x=i}^{N}\Big|P(x,y) - P(x,y+1)\Big| \tag{10}$$

We have considered that the SAD is a good measure on how much the pixels vary on a region and thus implemented the same concept in our prototype. The Python code for the function computing this value is:

```
1  def compute_SAD_for_block(block):
2      sad = 0
3
4      for i in range(0, BLOCK_ROWS-1):
5          for j in range(0, BLOCK_COLS-1):
6              sad += np.abs(block[i][j] - block[i+1][j]) + np.abs(block[i][j]
                   - block[i][j+1])
7      return sad
```

The above code takes a block of pixels of size BLOCK_ROWS x BLOCK_COLS as input and returns the sum of absolute differences in that block between neighbouring pixels. The function loops through the block starting from the upper left corner to the lower right one and at each step adds to the total SAD value the differences between the current pixel and the one on the right side and the one below.

The next step is for each block that is to be examined to construct a so called detection kernel [2], that consists of the neighbouring blocks. Ringing artifacts and mosquito noise appear and are visible to the human eye in areas of low intensity that contain also high intensity elements such as strong edges. Thus labeling a block as artifacted or not just by looking at its information is impossible if we don't consider the entire region to which it belongs. In the method of Eerenberg et al [2] the necessity of having both textural and flat blocks in the detection kernel was one of the core constraints of the spatial reasoning. For these reasons we have decided to adopt as well the idea of a detection kernel for identifying ringing and mosquito noise. Eerenberg et al [2] proposed a detection kernel of 5x3 blocks, but as there was no reasoning for using a asymmetrical one, we decided to use a symmetrical kernel. After some experiments conducted to identify what would an appropriate kernel size be, we have decided to stick to a 3x3 blocks dimension (Figure 11). A smaller one might not be able to grasp enough information about the surrounding region, while a higher one might increase the difficulty of spatial reasoning. Regarding the size of each block in particular, Eerenberg et al [2] mention a 5x5 pixels size and we adopted the same values.

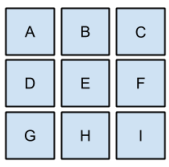


Figure 11: Ringing/mosquito noise detection kernel for a center block E

4.2.2 Spatial Reasoning

Eerenberg et al [2] introduced the term of spatial reasoning to refer to the process of analyzing the contamination of the central block in a detection kernel, by looking at the SAD values and checking if a set of boolean constraints is satisfied or not. A number of five thresholds was introduced to which the SAD values were supposed to be compared. The difficulty in setting out appropriate values for the thresholds and the ambiguity regarding the difference between some of them motivated us to simplify a little bit the entire spatial reasoning. Our approach takes the exact same processing steps as the one of Eeerenberg et al [2], but at each one of them introduces more simplicity that we believe can give at least similar results. We will present in the followings the steps of the spatial reasoning in its simplified version and with mentions to what we have changed compared to the method of Eerenberg et al [2].

Mosquito noise and ringing artifacts appear in areas where a transition from a flat area to a textural one is present. In other words appear next to the edges of elements that are placed on a smooth background. As a consequence, the presence of such an artifact in the central block of a detection kernel must be connected to the existence of flat and textural blocks in the surrounding blocks of the detection kernel.

In order to label a region as flat or textural we have considered two thresholds against which the SAD values would be compared, namely T_flat and T_tex. If a block would have a SAD value smaller than T_flat, then it would be considered flat and if it would have a value larger than T_tex, it would be considered textural. At the same time, the central block in the detection kernel can be potentially contaminated if it's neither flat, nor texture and thus its SAD value is somewhere in between the T_flat and T_tex. This gives a big simplification from the 5 thresholds proposed by Eerenberg et al [2] (Tl, T_flat, Tl_mos, Th_mos, Th) that also didn't have a clear differentiation.

By comparing the 9 blocks in the detection kernel against the two thresholds we have created two sets of 9-tuples (L and T) of boolean values, with the value 'true' if the corresponding block in the kernel is flat, respective textural and 'false' otherwise. In contrast, in the method of Eerenberg et al [2] a number of three 15-tuples sets were created.

$$\begin{aligned} F &= (f_A, f_B, f_C, f_D, f_E, f_F, f_G, f_H, f_I) \\ T &= (t_A, t_B, t_C, t_D, t_E, t_F, t_G, t_H, t_I) \end{aligned} \tag{11}$$

The next step is to apply a set of constraints that reflect the idea of having both flat and textural areas in the neighborhood of an artifact. We produced in this sense the following set of boolean equations:

$$\begin{aligned} &\text{flat\_top} = (f_A \wedge f_B) \vee (f_B \wedge f_C) \\ &\text{flat\_bottom} = (f_G \wedge f_H) \vee (f_H \wedge f_I) \\ &\text{flat\_left} = (f_A \wedge f_D) \vee (f_D \wedge f_G) \\ &\text{flat\_right} = (f_C \wedge f_F) \vee (f_F \wedge f_I) \\ &\text{flat} = \text{flat\_top} \vee \text{flat\_bottom} \vee \text{flat\_left} \vee \text{flat\_right} \\ &\text{tex} = t_A \vee t_B \vee t_C \vee t_D \vee t_F \vee t_G \vee t_H \vee t_I \\ &\text{centre} = \neg l_E \wedge \neg t_E \\ &\text{artifact} = \text{flat} \wedge \text{tex} \wedge \text{centre} \end{aligned} \tag{12}$$

The reasoning behind these equations is that in order to have a central artifacted block in the kernel, there needs to be at least one textural block (sharp transition) and a continuous flat surface of two blocks around it. At the same time, the central block needs to be none of the two, so its SAD value must have a value in between the T_flat and T_text. Compared to the set of equations introduced by Eerenberg et al [2], the constraint regarding textural blocks is the same, while the one for a flat surface has changed by allowing smaller continuous regions to satisfy the constraint.

The Python function that checks for a matrix of nine SAD values whether the mentioned constraints are satisfied or not is:

```
def check_if_artifacted(blocks_SADs):

    F = blocks_SADs < T_FLAT
    T = blocks_SADs > T_TEX

    flat_top = (F[0][0].all() and F[0][1].all()) \
        or  (F[0][1].all() and F[0][2].all())
    flat_bottom = (F[2][0].all() and F[2][1].all()) \
        or (F[2][1].all() and F[2][2].all())
    flat_left = (F[0][0].all() and F[1][0].all()) \
        or (F[1][0].all() and F[2][0].all())
    flat_right = (F[0][2].all() and F[1][2].all()) \
        or (F[1][2].all() and F[2][2].all())

    flat = flat_top or flat_bottom or flat_left or flat_right

    tex = False
    for i in range(0, len(T)):
        for j in range(0, len(T[i])):
            if i != 1 and j != 1:
                tex = tex or T[i][j].all()

    center = (T_FLAT < blocks_SADs[1][1]).all() \
        and (blocks_SADs[1][1] < T_TEX).all()

    artifacted = tex and flat and center
    return artifacted
```

The above function takes as an input a 3x3 matrix that contains the SAD values for the blocks in a detection kernel and returns a boolean value, stating whether the center block is or is not artifacted. First two boolean 9-tuples (F and T) are computed by comparing the SAD values to the thresholds T_FLAT and T_TEX. Further on, the existence of a flat sequence is evaluated at the top, the bottom, the left or the right side of the center block. For example, the flat_left boolean would have the value true if the first and second blocks in the first column are flat or if the second and the third blocks in the first column are flat. Afterwards, the existence of a textural block in the detection kernel is evaluated by looping through all the values in the T 9-tuple, except of the one corresponding to the center block. Finally, before returning the output, the function checks if the SAD value of the center block is between the T_FLAT and T_TEX thresholds.

The visualization of the ringing artifacts detection algorithm can be observed in Figure 12.

4.2.3 Quantification and Visibility Evaluation

Similar to blocking artifacts, quantifying the results of the detection algorithm is equally important to get a relevant measure. The process in the case of ringing and mosquito noise is fairly similar at its base, but with different detailed implementation.

The first measure that we are trying to get is the percentage of blocked regions with regards to the total number of regions in the image. As opposed to blocking artifacts, ringing and mosquito noise are prone to appear in images with a high percentage of sharp edges on smooth areas. The percentage of artifacted regions will be highly influenced by the structure of the image.

We have considered in this sense a new measure, based on the idea that not only the amount of ringing artifacts and mosquito noise is relevant, but also a measure of their overall visibility/annoyance should be computed [7]. The measure we use is different from the one proposed by Liu et al [7] and for each artifacted region in particular we compute a Mean of Absolute Differences, which is actually the previously calculated SAD value divided by the number of neighbouring pixel differences in the block. The annoyance score should emphasize on average how much neighbouring pixels inside a contaminated block vary, with small scores indicating a low visibility of the artifacts and high scores indicated annoying distortions.

(a)

(b)

Figure 12: Ringing and mosquito noise artifacts detection (a) stop.jpg image compressed at quality 60 and with visible ringing and mosquito noise artifacts; (b) Highlighting of the artifacts based on the results of the detection algorithm

5 Psychovisual human experiment

The experiment has been conducted as mentioned in section 3 to identify what users consider to be an acceptable quality level for a JPEG compressed image. The term of "psychovisual experiment" [7] has been identified during the literature survey and basically represents an user-based image quality evaluation. Mentions about psychovisual quality evaluators can be found also in the work of Chan et al [8].

5.1 Experiment setup

The setup of our experiment is totally different from the one of the experiment of Liu et al [7]. Our main goal was to use a large variety of images on different subjects and try to identify compression acceptability ranges for all of them. The main elements of the experiment are:

- 6 test subjects with different backgrounds (developers, designers, managers, video production specialists).
- a set of 60 images from different categories (photos from several settings, cliparts, drawings, small/large images, images with few/many colors, images with text, images with lines, images with gradients, screenshots etc.).
- users were presented with a slider and were asked to compress the images as far as they can by keeping the perceived quality at acceptable levels compared to the original one; the original and compressed images were presented next to each other on the same screen and the subjects were able to jump in steps of 5 between quality levels.
- the environment was fully controlled by having the same lightning for all subjects, same evaluation distance to the same screen, same screen resolution, black background to avoid distractions.
- the original images were either in the PNG format or in the JPEG format at a very high quality (95-100) that presented no visible artifacts; this choice has been made to allow the subject to compare compressed images to artifact-free ones, rather than to some that already presented distortions.

5.2 Results Analysis

The results of the experiment gave us for each image a compression range for which users consider the image to give acceptable compression results and they can be found in Appendix B.1.

The next step we wanted to take was to identify a central tendency for each of these ranges which would afterwards be the representative lowest quality level for each image where it's considered to be acceptable. The main difficulty that we have faced here proved to be the very large variations in quality ranges for the set of images. A small quality range was an indicator that all the subjects agreed on almost the same level of compression that gives an acceptable output for a specific image. In contrast, a large quality range could have been the consequence of several factors: different perceptions on what an acceptable quality represents, assessment of the quality based on different regions of the image, the differences in the subjects' background etc. Examples of high and small quality ranges are:

- Image 20.jpg - quality range 80-90
- Image 33.png - quality range 25-90

The choice of a representative value for these quality ranges was considered between the mean value, the median value and the value corresponding to a percentile of acceptability (e.g. 4 out of 6 subjects). The mean and median value were very close to each other for almost all the image, but as the ranges were often very large, we have decided that in the context of TinyJPG it would be more relevant a percentile of satisfied users and thus we adopted the same strategy for the current experiment. We considered a percentile of 83% (5 out of 6 subjects) to be a totally acceptable quality, while a percentile of 33% (2 out of 6 subjects) would be totally unacceptable.

6 Evaluation and Validation Results

The goal of the evaluation process was to identify whether the proposed measures of artifacts can be a good indicator for the quality of a JPEG compressed image. We have also analyzed how can this information be used in the encoding process.

6.1 Ringing and mosquito noise analysis

The first requirement in establishing whether the developed metrics are valuable or not was to check whether the artifact detection gives satisfying results. We have to mention that we considered that the ringing artifacts detection algorithm didn't reach our expectations. During the current research project four different approaches have been tried, starting from a personal implementation of the algorithm of Eerenberg et al [2], to a machine learning algorithm based on Support Vector Machines (SVM), a comparison-based method between the original and compressed image and the simplified version of Eerenberg et al [2] presented in the current paper. The first and last versions we consider give closely similar results, but still the number of false positive detections and misses makes us consider these approaches insufficient.

(a) (b) (c)

Figure 13: (a) Original image (b) Compressed image with slightly visible artifacts (c) Highlighting of artifacts based on the results of the detection algorithm with multiple false positives

(a) (b) (c)

Figure 14: (a) Original image (b) Compressed image with visible artifacts (c) Highlighting of artifacts based on the results of the detection algorithm with multiple misses

The main difficulty that we encountered in doing a block-based spatial detection of ringing artifacts consisted in reasoning about the types of areas in which they appear and also in setting up appropriate values for the thresholds towards which the SAD values are compared. We have identified examples of

images that generate false positives due the similarity of the SAD values of an artifacted region and a textural region (e.g. Figure 13). Additionally we have identified example images that generate a lot of misses due to the invalidity of some of the spatial reasoning constraints in the method of Eerenberg et al [2]. Such an example can be observed in Figure 14 where an important fraction of the visible artifacts are not detected. We have tried to solve these aspects in the presented simplified version, but still we haven't reached the desired outcome. We thus consider that further work needs to be carried out in order to reach a more accurate detection, if actually possible. Only afterwards, the ringing artifact measures can be evaluated in the context of quantifying the appearance of artifacts.

Considering these, we will focus in the followings on evaluating the measures that we have developed for blocking artifacts as the detection algorithm we consider gives satisfying results in this case.

6.2 Blocking artifacts analysis

During the psychovisual experiment we have realized that the images degrade from different points of view based on their structure. Images that contain large smooth areas with low differences in gradient are usually prone to blocking artifacts, images that have strong edges are prone to a degradation due to ringing and mosquito noise artifacts, while other images degrade due to a loss of color, contrast, introduced bluriness etc. Examples of such images can be seen in Appendix B.2. Based on these findings, we have decided to evaluate if the proposed metrics for blocking artifacts are a good perceived quality indicator by looking only at images were the subject would decide mostly by the quality of smooth areas, so by the appearance of the blocking artifacts.

We have thus selected the only 8 images out of the set of 60 we have used in the experiment described in section 5 that contained mostly smooth areas. The next step was to detect and measure the blocking artifacts on the 8 images for qualities that are totally acceptable (83% acceptability percentile) and higher (100% acceptability percentile), and as well for totally unacceptable qualities (33% acceptability percentile) and lower (16% acceptability percentile).

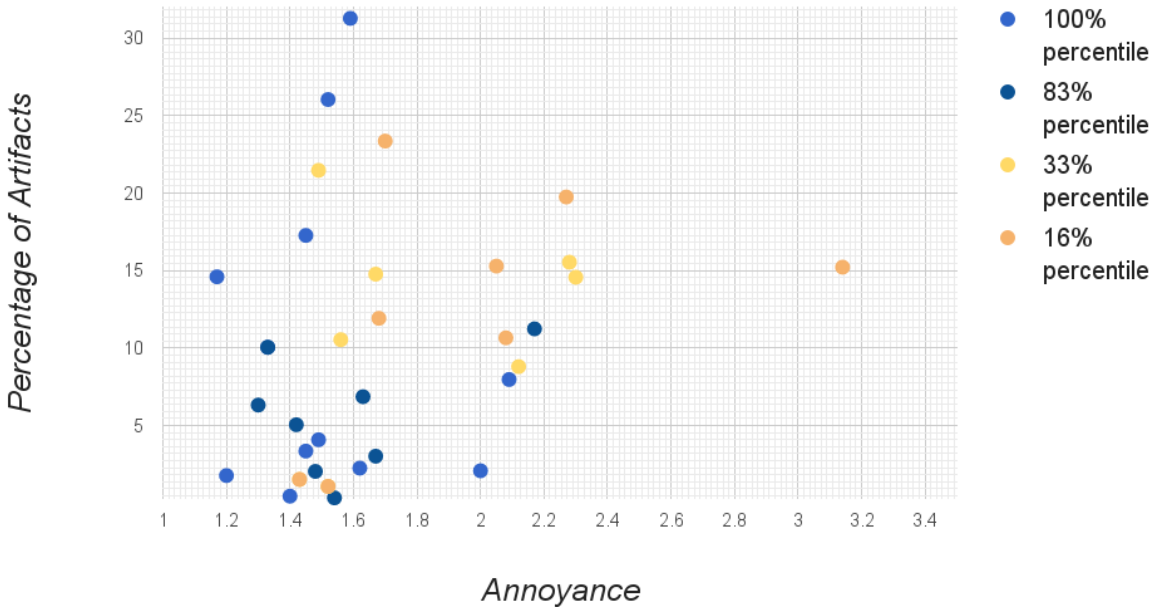


Figure 15: Scatter plot of blocking artifacts measurements

Figure 15 shows a scatter plot of the measurement of blocking artifacts on the 8 representative images. Based on this visualization we have observed a clustering trend of the points representing acceptable and on the other side unacceptable qualities, with the presence also of a couple of outliers. We have tried in this sense to draw a separation line between the two clusters. The approach we took in identifying this

line was to first compute the centroids of the two clusters, join them by a line and finally draw through the middle of this line a perpendicular one that would represent the desired separation line. The same plot including also the separation line can be observed in Figure 16, while the equation of the line is:

$$y = -20.9x + 43.9 \tag{13}$$

This aspect indicated us that there could exist a linear relation separating acceptable images from unacceptable ones. At this point we formulated the hypothesis stating that the identified separation line can be a good indicator on the acceptability of an image highly dominated by smooth areas, from the point of view of blocking artifacts.

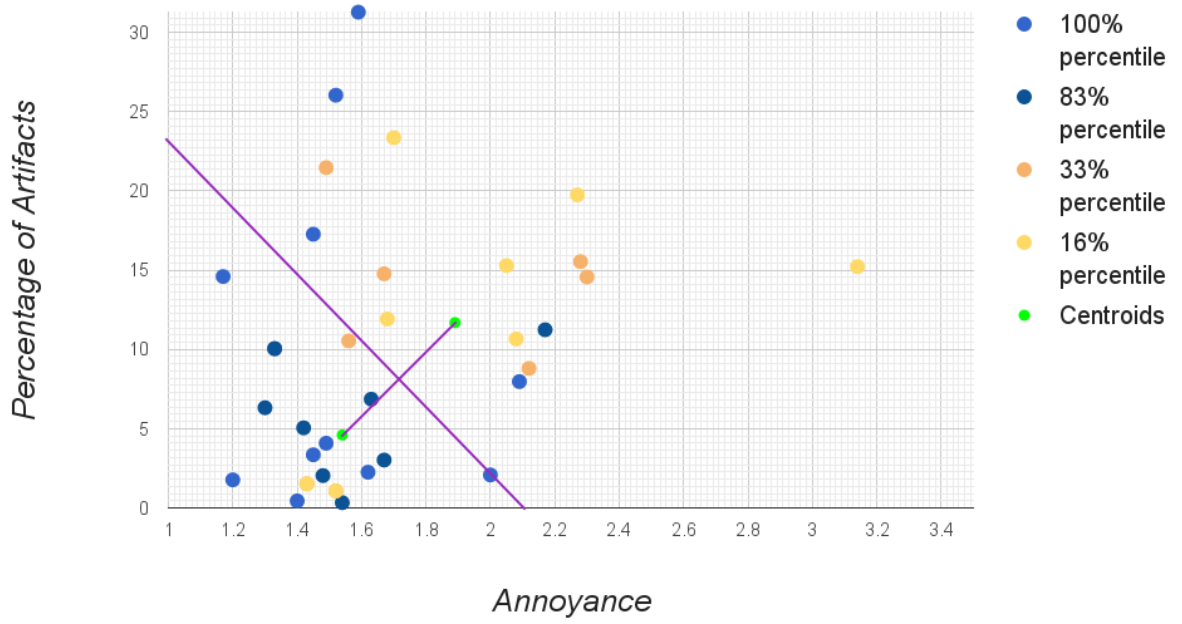


Figure 16: Scatter plot of blocking artifacts measurements and the separation line between the clusters of acceptable and unacceptable qualities

The next step was to verify this hypothesis by conducting another experiment. We organized a validation experiment that had an exactly similar setup to the psychovisual experiment, the only difference being the set of images. We built up a set of 5 images all consisting of mostly smooth areas prone to blocking artifacts. Afterwards we did identical measurements on different levels of acceptability of the 5 images and the scatter plot of results can be observed in Figure 17. The evaluation of these results shows that for 90% of the data points the formulated hypothesis is valid. We have thus considered that the identified separation line can actually be a good indicator on the perceived acceptable quality of a compressed image highly dominated by smooth areas.

6.3 Discussion

The current implementation of TinyJPG uses a similarity metric between the original and compressed image to decide on how far the compression rates should go. We have been investigating in the current project if a measurement of the compression artifacts could also be a good predictor for the highest compression that still gives acceptable results from the point of view of the human visual perception.

We have validated in our research process a hypothesis stating that measures of blocking artifacts can be a good indicator on the acceptability of an image, but only if the image is highly dominated by smooth regions. Images that have more textural elements (e.g. objects) with sharp edges are prone to ringing artifacts. On such images our blocking artifacts evaluator might asses the images as acceptable at some

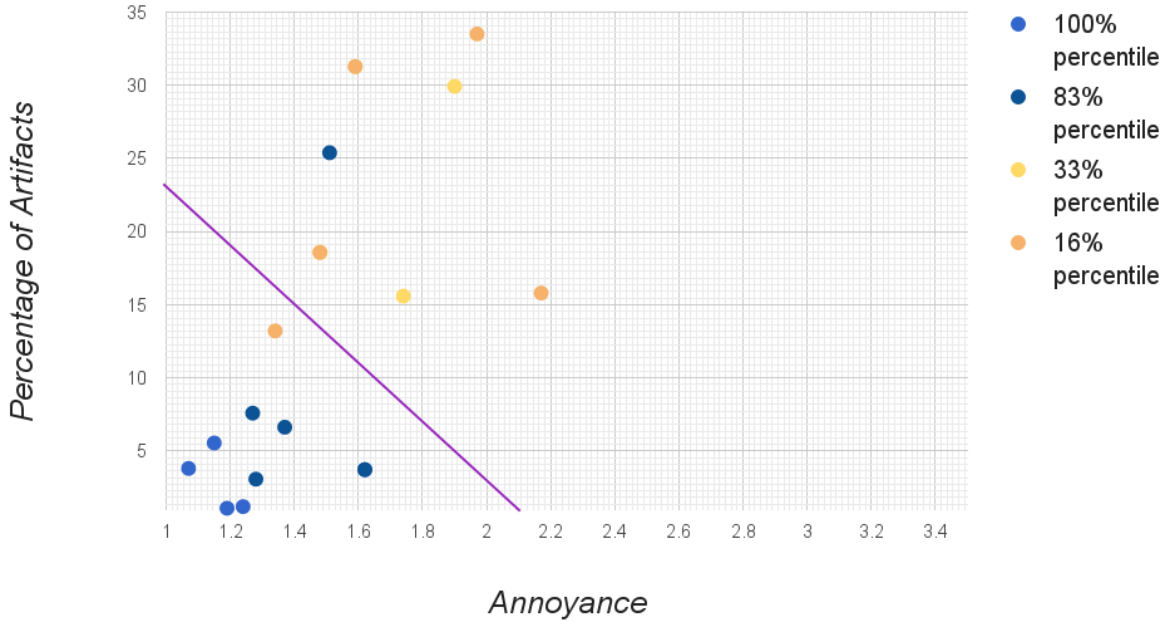


Figure 17: Scatter plot of the validation results

low qualities even though they are totally unacceptable due to the presence of ringing artifacts. We thus consider that the existence of a ringing artifacts evaluator is totally necessary if a larger variety of images needs to be taken into consideration.

The results of the psychovisual experiment also gave us an idea that sometimes the image degradation might not be due to the introduction of blocking, ringing or mosquito noise, but can have other causes such as a loss of color, contrast, introduced bluriness etc. For such reason, the artifact measures might still label a compressed image as acceptable even though its quality decreased a lot in other directions. We have thus concluded that measures of blocking artifacts can for sure not replace a similarity metric in deciding when to stop compressing. The two however could both be taken into consideration into the decision making process and we consider this as possible further work.

To summarize, the answers for the proposed research questions would be:

- Can we measure how much a JPEG compressed image is affected by artifacts?

  Yes, we were able to quantify the appearance of blocking artifacts through two different measures, the total percentage of artifacts and an annoyance score. However, in the case of ringing and mosquito noise artifacts the detection system needs a different approach for more accurate results and only afterwards the measurement of artifacts can be considered.

- Can the quantification of artifacts (their amount, visibility estimation) be a good indicator on the acceptability of a compressed image?

  Yes, in the case of blocking artifacts we have proved that the selected measures (percentage of artifacts and annoyance score) can be a good indicator on the acceptability of a compressed image.

- Can this information be used to improve on existing techniques that decide on the quality of a compressed image by using similarity metrics?

  We consider that measures of artifacts can't replace similarity metrics in deciding on the quality of a compressed image. Taking both into consideration could bring improvements, but that evaluation can only be done when having also a proper ringing detection.

The software developed during the research project consisted of several Python scripts that represented prototypes of blocking and ringing artifacts detection and measuring algorithms. The idea was to use

a programming language that has an ease of prototyping and not necessarily one that would generate production code. The current implementation of TinyJPG is written in C++, but for the above mentioned reason we have chosen Python. We wanted to research the idea of using the information of the JPEG artifacts in the compression process and in case of promising results actually integrate it into TinyJPG.

Regarding the detection algorithms, one version has been developed for blocking artifacts and four different versions for ringing artifacts, out of which the one that gave the best results has been presented in this paper. The corresponding code for the methods presented in this paper can be found in Appendix A.

We see the integration of the developed algorithms into the TinyJPG project possible at least through two different approaches. The main advantage that would make this integration easy, is the independence of the algorithms. Both the blocking and the ringing algorithms only need the current output of the TinyJPG compression to analyze the image and decide on its acceptability. The first approach would be a conversion of the Python code into C++ code. The main libraries used in the developed Python scripts are OpenCV and NumPy, the first one facilitating the conversion as it is supported also in C++. Regarding NumPy, a corresponding C++ library for scientific computing that offers similar functionality needs to be used. The second approach would imply a direct integration of the Python scripts without any conversion to C++. However Python is not currently used in TinyJPG and thus with this option, the only advantage would be a possibly faster integration.

6.4 Threats to validity

The design of the conducted experiments imposes a couple of threats to its validity. We will shortly analyze some of the main aspects that could have influenced the results.

The subjects. The background of the subjects and their connections to TinyJPG could have influenced the results. Some of the subjects have been actively involved in the development of TinyJPG and thus were more strict regarding what an acceptable quality of a compressed image represents. This has been observed also from the results, where the subject with the highest average quality choice is one of the main developers of TinyJPG. In contrast, subjects that had no involvement with the compression tool were more likely to allow lower qualities.

A second aspect regarding the choice of subjects is connected to the fact that we have used the same users both for the first experiment based on which we created our hypothesis and for the validation experiment. It can be the case that at least the prediction accuracy of the separation could have been different with a different group of subjects completing the validation experiment.

The images. The choice of images is highly important in such an experiment. We have tried to cover as many categories of images as possible, but it is of course possible that the results would have been different with other image samples. Additionally, some types of images could still have been missed.

The subjects' focus and attention to detail. We have observed during the experiment that subjects analyze images from different points of view, they look at different areas of the images when deciding on the acceptability. This can thus influence their choices as distortions can be missed even if present. A second observed aspect was the speed of analyzing an image where subjects again took different amounts of time. The completion time of the first experiment varied between 15 and 30 minutes for analyzing the 60 images in the set. The two mentioned aspects could perhaps have been controlled even more by giving the users a preset time frame to analyze each image.

The setup conditions. The variables that we have controlled regarding the environmental setup were the lightning, the viewing distance and angle, the screen resolution. We consider that changing any of these variables would influence the results.

The quality evaluation step. The subjects were able to compare images between quality levels 0 and 100, with a step of 5. We consider that for some image their appearance doesn't change that much between two consecutive steps (e.g. quality 75 and 80) and thus the choice of an acceptable quality was difficult for the subjects. In case of redoing the same experiment with the same subjects the results might still slightly vary.

6.5 Future work

Measures of blocking artifacts gave us very promising results when estimating the quality of an image that is highly dominated by smooth areas. In our view, measures of ringing artifacts can be equally valuable for images containing sharp edges. For this reason, we would like to further investigate other approaches of detecting ringing artifacts that could give much more accurate results and thus measurements could be carried out as well.

Afterwards, in the context of TinyJPG, an analysis could be done on whether using the knowledge of artifacts together with the similarity metric can give better acceptability estimations than using only a similarity metric.

7 Conclusion

During the current research project we have analyzed whether measures of compression artifacts in a JPEG image can be a good indicator on the quality acceptability of the image. We have developed in this sense two algorithms for detecting the most common types of artifacts, blocking on one side and ringing and mosquito noise on the other side. Furthermore we have tried to quantify the appearance of these artifacts through different measures that could afterwards be possible quality indicators.

Measures of blocking artifacts proved to be a very good indicator on the acceptability of an image with a 90% prediction accuracy. However these type of artifacts are prone to images that contain mostly smooth areas and only for such images their measures can be a good indicator.

Regarding ringing and mosquito noise artifacts, the implemented detection method did not offer the expected performance and we consider that is because of the difficulty of covering all the possible spatial circumstances under which these artifacts can appear. Due to this reason, an evaluation on whether a measure of the detected artifacts can be a good indicator on the quality of the image was not possible anymore. We strongly consider that other alternatives for detecting ringing and mosquito noise artifacts are worth investigating as their measures might also be a good indicator of image acceptability where blocking artifacts measures are not.

Finally, we have concluded that measuring artifacts on a JPEG compressed image however isn't enough for deciding on its acceptability. We have observed that some images simply degrade from other points of view and thus artifacts measures should only be used together with other metrics.

Acknowledgements

I would like to express my gratitude to the Voormedia team that created the perfect environment for me to conduct my research and especially to Jacob, Jippe and Rolf that offered me continuous help in fulfilling my goals. At the very same time I would like to thank to my supervisor, dr. Magiel Bruntink, for his permanent guidance and support during all this process. I would also like to thank to dr. Rein van den Boomgaard for the helpful mentoring sessions. Finally, I would like to especially thank my parents who support me in everything I'm doing and are always there for me.

References

[1] Gregory K. Wallace. *The JPEG Still Picture Compression Standard*, in IEEE Transactions on Consumer Electronics, December 1991

[2] Onnon Eerenberg, Jeroen Kettenis, Peter H.N. de With. *Block-Based Detection Systems for Visual Artifact Location*, in IEEE International Conference on Consumer Electronics (ICCE), 2013

[3] Shigenobu Minamo, Avideh Zakhor. *An Optimization Approach for Removing Blocking Effects in Transform Coding*, in IEEE Transactions on Circuits and Systems for Video Technology, Vol. 5, No. 2, April 1995

[4] Gopal Lakhani, Norman Zhong. *Derivation of Prediction Equations for Blocking Effect Reduction*, in IEEE Transaction on Circuits and Systems for Video Technology, Vol. 9, No. 3, April 1999

[5] George A. Triantafyllidis, Dimitros Tzovaras, Michael Gerassimos Strintzis. *Blocking Artifact Detection and Reduction in Compressed Data*, in IEEE Transactions on Circuits and Systems for Video Technology, Vol. 12, No. 10, October 2002

[6] Hantao Liu, Nick Klomp, Ingrid Heynderickx. *Perceptually relevant ringing region detection method*, 16th European Signal Processing Conference (EUSIPCO 2008), Lausanne, Switzerland, August 2008

[7] Hantao Liu, Nick Klomp, Ingrid Heynderickx. *A No-reference Metric for Perceived Ringing*, in IEEE Transaction on Circuits and Systems for Video Technology, Vol. 20, No. 4, April 2010

[8] Ruby Wai-Shan Chan, Peter Goldsmith. *Modeling and validation of a psychovisually based image quality evaluator for DCT-based compression*, in Signal Processing: Image Communication, Vol. 17, pp. 485-495, 2002

[9] Irina Popovici, Wm. Douglas Withers. *Locating Edges and Removing Ringing Artifacts in JPEG Images by Frequency-Domain Analysis*, in IEEE Transactions on Image Processing, Vol. 16, No. 5, May 2007

[10] Ying Luo, Rabab K. Ward. *Removing the Blocking Artifacts of Block-Based DCT Compressed Images*, in IEEE Transactions on Image Processing, Vol. 12, No. 7, July 2003

[11] Yuhong Zhang, Ezzatollah Salari, Shuangteng Zhang. *Reducing blocking artifacts in JPEG-compressed images using an adaptive neural-network algorithm*, in Neural Computing and Applications, Vol. 22, pp. 3-10, January 2013

[12] Mei-Yin Shen, C.-C. Jay Kuo *Review of Postprocessing Techniques for Compression Artifact Removal*, in Journal of Visual Communication and Image Representation, Vol. 9, pp. 2-14, 1998

[13] Fang Zhu. *Edge Map Based Blocking Artifacts Reduction*, in 2009 International Conference on Computer Engineering and Applications, pp. 693-697

[14] Yanhui Xia, Zhengyou Wang, Wan Wang, Jin Wang, Zheng Wan, Weisi Lin. *Blind Measurement of Blocking Artifacts of Images Based on Edge and Flat-region Detection*, in Journal of Software, Vol. 8, No. 1, pp. 168-175, January 2013

[15] Huibin Chang, Michael K. Ng, Tieyong Zeng. *Reducing Artifact in JPEG Decompression via a Learned Dictionary*, in IEEE Transactions on Signal Processing, Vol. 62, pp. 718-728, November 2013

[16] Sangjin Kim, Sinyoung Jun, Eunsung Lee, Jeongho Shin, Joonki Paik. *Ringing Artifact Removal in Digital Restored Images Using Multi-Resolution Edge Map*, in International Journal of Signal Processing, Image Processing and Pattern Recognition, Vol. 2, No. 4, December 2009

A Artifact Detection and Measuring Algorithms

A.1 Blocking Artifacts

```
1  import numpy as np
2  import cv2
3  import subprocess
4
5  BLOCK_ROWS = 8
6  BLOCK_COLS = 8
7  HORIZONTAL_DIRECTION = 1
8  VERTICAL_DIRECTION = 2
9
10 BLOCKINESS_LOW_THRESHOLD = xxx
11 BLOCKINESS_HIGH_THRESHOLD = xxx
12
13 class ArtifactedEdge:
14     point1 = None
15     point2 = None
16     annoyance = None
17
18     def __init__(self, point1, point2, annoyance):
19         self.point1 = point1
20         self.point2 = point2
21         self.annoyance = annoyance
22
23 def highlight_image_artifacts(image, artifacted_edges):
24     for edge in artifacted_edges:
25         cv2.line(image, edge.point1, edge.point2, (0, 0, 0))
26
27 def compute_overall_annoyance(artifacted_edges):
28     annoyance = 0
29
30     if len(artifacted_edges) != 0:
31         for edge in artifacted_edges:
32             annoyance += edge.annoyance
33         return annoyance/len(artifacted_edges)
34     else:
35         return 0
36
37 def compute_edge_annoyance(first_block, second_block, direction):
38     if direction == VERTICAL_DIRECTION:
39         return np.average(np.abs(second_block[0:1, 0:BLOCK_COLS] -
               first_block[BLOCK_ROWS-1:BLOCK_ROWS, 0:BLOCK_COLS]), axis=1)
40
41     if direction == HORIZONTAL_DIRECTION:
42         return np.average(np.abs(second_block[0:BLOCK_ROWS, 0:1] -
               first_block[0:BLOCK_ROWS, BLOCK_COLS-1:BLOCK_COLS]), axis=0)
43
44 def has_low_pixel_variation(pixel, pixel_array, diff):
45     for x in pixel_array:
46         current_diff = np.abs(pixel - x)
47         if not (np.greater_equal(current_diff, diff-3).all() \
48             and np.greater_equal(diff+3, current_diff).all()):
49             return False
50     return True
51
52 def check_blockiness(first_block, second_block, direction):
53     total_blockiness = 0
54     size = len(first_block)
```

```
55      blockinesses = []
56
57      for x in range(0, size):
58          current_blockiness = 0
59          if direction == VERTICAL_DIRECTION:
60              boundary_slope = np.abs(second_block[0][x] - first_block[size
                    -1][x])
61              if not has_low_pixel_variation(first_block[size-1][x],
                    second_block[0:size-1, x:x+1], boundary_slope) \
62                      or not has_low_pixel_variation(second_block[0][x],
                            first_block[0:size-1, x:x+1], boundary_slope):
63                  return False
64              first_slope = np.abs(first_block[size-1][x] - first_block[size -
                     2][x])
65              second_slope = np.abs(second_block[1][x] - second_block[0][x])
66              current_blockiness = boundary_slope - np.float_(first_slope +
                    second_slope)/2
67          elif direction == HORIZONTAL_DIRECTION:
68              boundary_slope = np.abs(second_block[x][0] - first_block[x][size
                    -1])
69              if not has_low_pixel_variation(first_block[x][size-1],
                    second_block[x:x+1, 0:size-1], boundary_slope) \
70                      or not has_low_pixel_variation(second_block[x][0],
                            second_block[x:x+1, 0:size-1], boundary_slope):
71                  return False
72              first_slope = np.abs(first_block[x][size-1] - first_block[x][
                    size-2])
73              second_slope = np.abs(second_block[x][1] - second_block[x][0])
74              current_blockiness = boundary_slope - np.float_(first_slope +
                    second_slope)/2
75
76          if np.greater(BLOCKINESS_LOW_THRESHOLD, np.float_(current_blockiness
                )).all() \
77                  or np.greater(np.float_(current_blockiness),
                        BLOCKINESS_HIGH_THRESHOLD).all():
78              return False
79
80          total_blockiness += current_blockiness
81          blockinesses.append(current_blockiness)
82
83      total_blockiness = np.float_(total_blockiness)/np.float_(size)
84
85      for b in blockinesses:
86          if np.greater(np.abs(total_blockiness - b), 2).any():
87              return False
88
89      blocked = (BLOCKINESS_LOW_THRESHOLD <= total_blockiness[0] <=
            BLOCKINESS_HIGH_THRESHOLD \
90              and total_blockiness[1] <= BLOCKINESS_HIGH_THRESHOLD \
91              and total_blockiness[2] <= BLOCKINESS_HIGH_THRESHOLD) \
92               or (BLOCKINESS_LOW_THRESHOLD <= total_blockiness[1] <=
                    BLOCKINESS_HIGH_THRESHOLD \
93                   and total_blockiness[0] <= BLOCKINESS_HIGH_THRESHOLD \
94                   and total_blockiness[2] <= BLOCKINESS_HIGH_THRESHOLD) \
95               or (BLOCKINESS_LOW_THRESHOLD <= total_blockiness[2] <=
                    BLOCKINESS_HIGH_THRESHOLD \
96                   and total_blockiness[0] <= BLOCKINESS_HIGH_THRESHOLD \
97                   and total_blockiness[1] <= BLOCKINESS_HIGH_THRESHOLD)
98
99      return blocked
100
```

```
101 def get_artifacted_edges(blocks):
102     artifacted_edges = []
103     for i in range(0, len(blocks) - 1):
104         for j in range(0, len(blocks[i]) - 1):
105             right_blocked = check_blockiness(blocks[i][j], blocks[i][j+1],
                    HORIZONTAL_DIRECTION)
106             if right_blocked:
107                 annoyance = compute_edge_annoyance(blocks[i][j], blocks[i][j
                        +1], HORIZONTAL_DIRECTION)
108                 artifacted_edges.append(ArtifactedEdge(((j+1)*BLOCK_COLS, i*
                        BLOCK_ROWS), ((j+1)*BLOCK_COLS, (i+1)*BLOCK_ROWS),
                        annoyance))
109
110             bottom_blocked = check_blockiness(blocks[i][j], blocks[i+1][j],
                    VERTICAL_DIRECTION)
111             if bottom_blocked:
112                 annoyance = compute_edge_annoyance(blocks[i][j], blocks[i
                        +1][j], VERTICAL_DIRECTION)
113                 artifacted_edges.append(ArtifactedEdge((j*BLOCK_COLS, (i+1)*
                        BLOCK_ROWS), ((j+1)*BLOCK_COLS, (i+1)*BLOCK_ROWS),
                        annoyance))
114
115     return artifacted_edges
116
117 def get_image_blocks(image):
118     blocks = []
119     rows, cols, ch = image.shape
120
121     for i in xrange(0, rows/BLOCK_ROWS):
122         blocks.append([])
123         for j in xrange(0, cols/BLOCK_COLS):
124             blocks[i].append(image[i*BLOCK_ROWS:(i+1)*BLOCK_ROWS, j*
                    BLOCK_COLS:(j+1)*BLOCK_COLS])
125
126     return blocks
127
128 def measure_artifacts(image_path, output_path):
129     image = cv2.imread(image_path, 1)
130     image_array = np.array(image, dtype=np.int64)
131
132     rows, cols, ch = image.shape
133
134     blocks = get_image_blocks(image_array)
135     artifacted_edges = get_artifacted_edges(blocks)
136
137     annoyance_score = np.average(compute_overall_annoyance(artifacted_edges)
            )
138     print 'Annoyance Score: %0.2f' % annoyance_score
139
140     total_artifacts_percentage = np.float_(len(artifacted_edges)) / np.
            float_(((rows/BLOCK_ROWS)*(cols/BLOCK_COLS)*2)) * 100
141     print 'Artifacted Edges: %0.2f %%' % total_artifacts_percentage
142
143     highlight_image_artifacts(image, artifacted_edges)
144     cv2.imwrite(output_path, image)
145
146     return (total_artifacts_percentage, annoyance_score)
```

A.2 Ringing and Mosquito Noise Artifacts

```
import numpy as np
import cv2
import cv
import math
import subprocess

BLOCK_ROWS = 5
BLOCK_COLS = 5
KERNEL_X = 3
KERNEL_Y = 3

T_FLAT = xxx
T_TEX = xxx

class ArtifactedBlock:
    x = None
    y = None
    annoyance = None

    def __init__(self, x_coord, y_coord, annoyance):
        self.x = x_coord
        self.y = y_coord
        self.annoyance = annoyance

def highlight_image_artifacts(image, artifacted_blocks):
    for block in artifacted_blocks:
        cv2.rectangle(image, (block.y*BLOCK_COLS, block.x*BLOCK_ROWS), ((
            block.y+1)*BLOCK_COLS, (block.x+1)*BLOCK_ROWS), (0,0,0))

def compute_overall_annoyance(artifacted_blocks):
    annoyance = 0

    if len(artifacted_blocks) != 0:
        for block in artifacted_blocks:
            annoyance += block.annoyance
        return annoyance/len(artifacted_blocks)
    else:
        return 0

def check_if_artifacted(blocks_SADs):

    F = blocks_SADs < T_FLAT
    T = blocks_SADs > T_TEX

    flat_top = (F[0][0].all() and F[0][1].all()) or (F[0][1].all() and F
        [0][2].all())
    flat_bottom = (F[2][0].all() and F[2][1].all()) or (F[2][1].all() and F
        [2][2].all())
    flat_left = (F[0][0].all() and F[1][0].all()) or (F[1][0].all() and F
        [2][0].all())
    flat_right = (F[0][2].all() and F[1][2].all()) or (F[1][2].all() and F
        [2][2].all())

    flat = flat_top or flat_bottom or flat_left or flat_right

    tex = False
    for i in range(0, len(T)):
        for j in range(0, len(T[i])):
            if i != 1 and j != 1:
```

```
55                  tex = tex or T[i][j].all()
56
57      centre = (T_FLAT < blocks_SADs[1][1]).all() and (blocks_SADs[1][1] <
            T_TEX).all()
58
59      artifacted = tex and flat and centre
60
61      return artifacted
62
63  def check_artifacted_blocks(blocks_SADs_map):
64      artifacted_blocks = []
65      for i in range(1, len(blocks_SADs_map) - 1):
66          for j in range(1, len(blocks_SADs_map[i]) - 1):
67              if check_if_artifacted(blocks_SADs_map[i-1:i+2, j-1:j+2]):
68                  annoyance = blocks_SADs_map[i][j]/((BLOCK_COLS-1)*(
                        BLOCK_ROWS-1)*2)
69                  artifacted_blocks.append(ArtifactedBlock(i,j,annoyance))
70
71      return artifacted_blocks
72
73  def compute_SAD_for_block(block):
74      sad = 0
75      for i in range(0, BLOCK_ROWS-1):
76          for j in range(0, BLOCK_COLS-1):
77              sad += np.abs(block[i][j] - block[i+1][j]) + np.abs(block[i][j]
                    - block[i][j+1])
78
79      return sad
80
81  def compute_blocks_SAD(blocks):
82      blocks_SADs = np.array([[[0,0,0] for x in range(len(blocks[0]))] for x
            in range(len(blocks))])
83      for i in range(0, len(blocks)):
84          for j in range(0, len(blocks[i])):
85              blocks_SADs[i][j] = compute_SAD_for_block(blocks[i][j])
86
87      return blocks_SADs
88
89  def get_image_blocks(image, rows, cols):
90      blocks = []
91
92      for i in xrange(0, rows/BLOCK_ROWS):
93          blocks.append([])
94          for j in xrange(0, cols/BLOCK_COLS):
95              blocks[i].append(image[i*BLOCK_ROWS:(i+1)*BLOCK_ROWS, j*
                    BLOCK_COLS:(j+1)*BLOCK_COLS])
96
97      return blocks
98
99  def measure_artifacts(image_path, output_path):
100     image = cv2.imread(image_path, 1)
101     image_array = np.array(image, dtype=np.int64)
102     print "reading image " + image_path
103
104     rows, cols, ch = image.shape
105
106     blocks = get_image_blocks(image_array, rows, cols)
107
108     blocks_SADs = compute_blocks_SAD(blocks)
109
110     artifacted_blocks = check_artifacted_blocks(blocks_SADs)
```

```
111
112         annotation_score = np.average(compute_overall_annoyance(artifacted_blocks
                ))
113         print 'Annoyance Score: %0.2f' % annoyance_score
114
115         total_artifacts_percentage = np.float_(len(artifacted_blocks)) / np.
                float_((rows/BLOCK_ROWS)*(cols/BLOCK_COLS)) * 100
116         print 'Artifacted Edges: %0.2f %%' % total_artifacts_percentage
117
118         highlight_image_artifacts(image, artifacted_blocks)
119         cv2.imwrite(output_path, image)
120
121         return (total_artifacts_percentage, annoyance_score)
```

B Pyschovisual Human Experiment

B.1 Results

The results of the experiment highlight the quality level that each subject has chosen for the 60 images as being the lowest one that still gives acceptable results in terms of perceived quality.

Image	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5	Subject 6
1.jpg	80	75	45	65	65	80
2.jpg	75	60	65	45	80	60
3.png	85	85	80	80	90	70
4.jpg	60	30	50	45	60	45
5.jpg	50	25	35	45	35	45
6.jpg	55	40	35	55	50	25
7.jpg	35	20	15	30	35	25
8.jpg	65	35	40	50	40	30
9.jpg	55	54	70	40	70	30
10.jpg	55	29	80	50	65	35
11.jpg	75	49	70	50	50	65
12.jpg	55	19	40	35	27	25
13.jpg	40	15	40	25	30	30
14.jpg	45	25	50	50	25	30
15.jpg	70	40	50	25	75	35
16.jpg	70	45	55	70	85	50
17.jpg	70	50	75	65	70	55
18.jpg	65	55	70	45	70	65
19.jpg	80	70	75	80	85	85
20.jpg	85	80	85	80	85	90
21.jpg	76	95	85	75	85	90
22.png	91	75	65	75	85	85
23.jpg	40	15	15	25	20	10
24.jpg	50	40	70	50	70	35
25.jpg	80	60	65	70	65	55
26.jpg	75	75	65	75	85	70
27.jpg	45	25	35	50	35	40
28.jpg	90	90	75	85	85	65
29.jpg	65	60	50	60	45	35
30.jpg	50	35	50	65	55	65
31.jpg	65	60	40	70	70	65
32.png	70	50	50	60	70	40
33.png	75	25	65	60	90	40
34.png	60	40	45	45	60	35
35.jpg	80	60	75	70	75	55
36.jpg	60	40	30	45	40	30
37.jpg	65	50	40	55	65	40
38.jpg	35	25	40	55	45	20
39.png	75	55	50	75	75	60
40.jpg	65	25	30	60	75	40
41.jpg	40	55	20	30	35	20
42.jpg	85	90	80	90	95	65
43.jpg	25	15	15	25	15	25
44.png	85	50	35	80	75	55
45.png	60	45	45	50	55	35
46.png	55	40	50	55	60	34
47.png	65	45	45	65	75	54
48.jpg	50	30	35	50	45	30

Image	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5	Subject 6
49.png	90	90	70	90	65	60
50.png	50	20	25	40	30	35
51.png	70	25	35	50	60	35
52.jpg	75	80	90	70	90	85
53.jpg	60	35	60	30	70	55
54.jpg	80	50	50	65	80	55
55.png	80	50	60	75	85	50
56.jpg	45	30	85	65	45	40
57.jpg	60	30	40	35	50	30
58.jpg	70	70	70	80	75	70
59.jpg	90	60	50	80	65	70
60.jpg	45	15	25	45	20	25

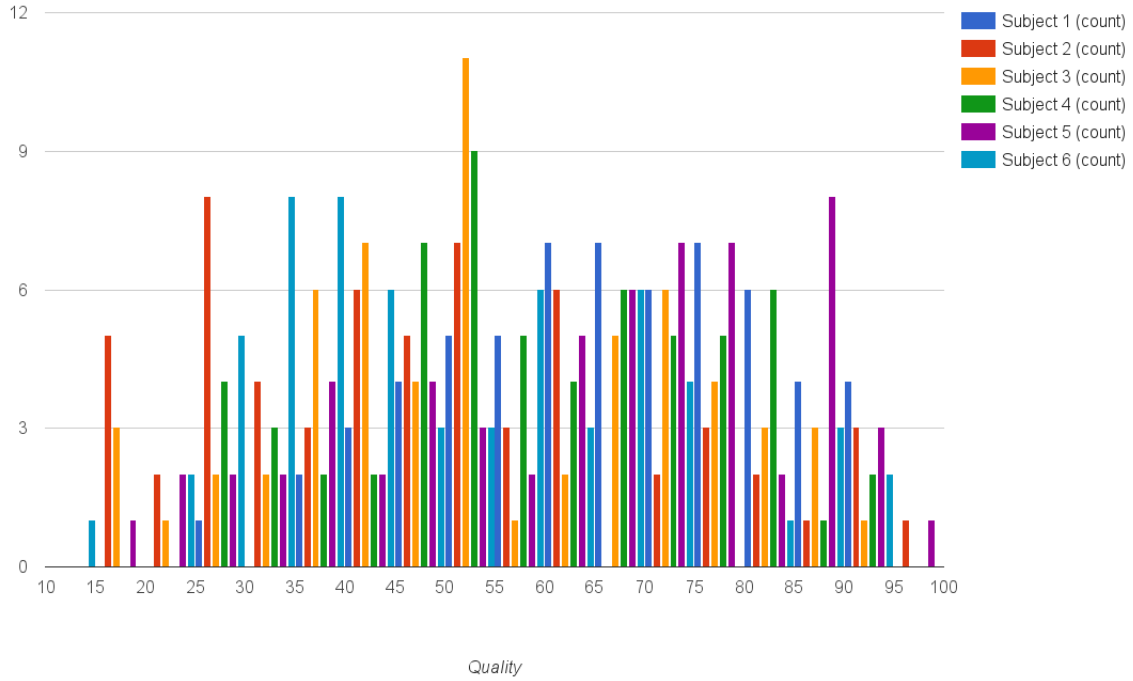
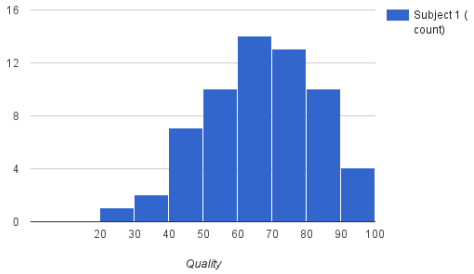
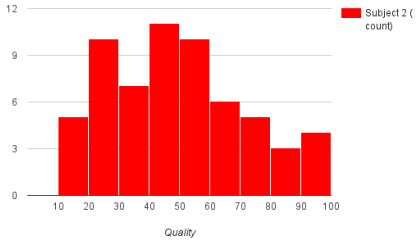


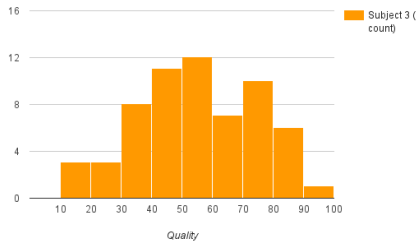
Figure 18: Quality choices distribution for all subjects



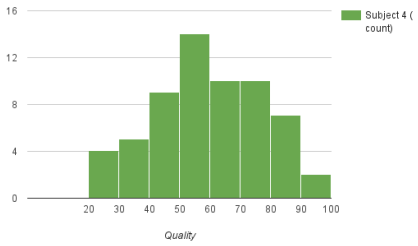
(a) Subject 1



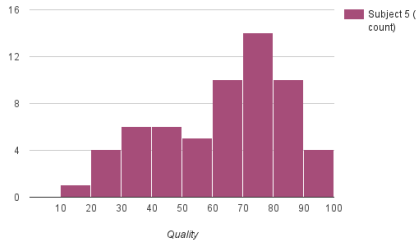
(b) Subject 2



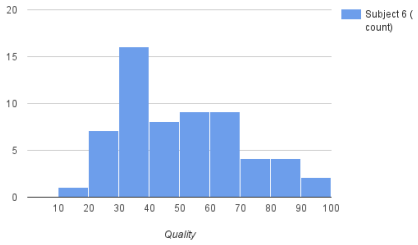
(c) Subject 3



(d) Subject 4



(e) Subject 5

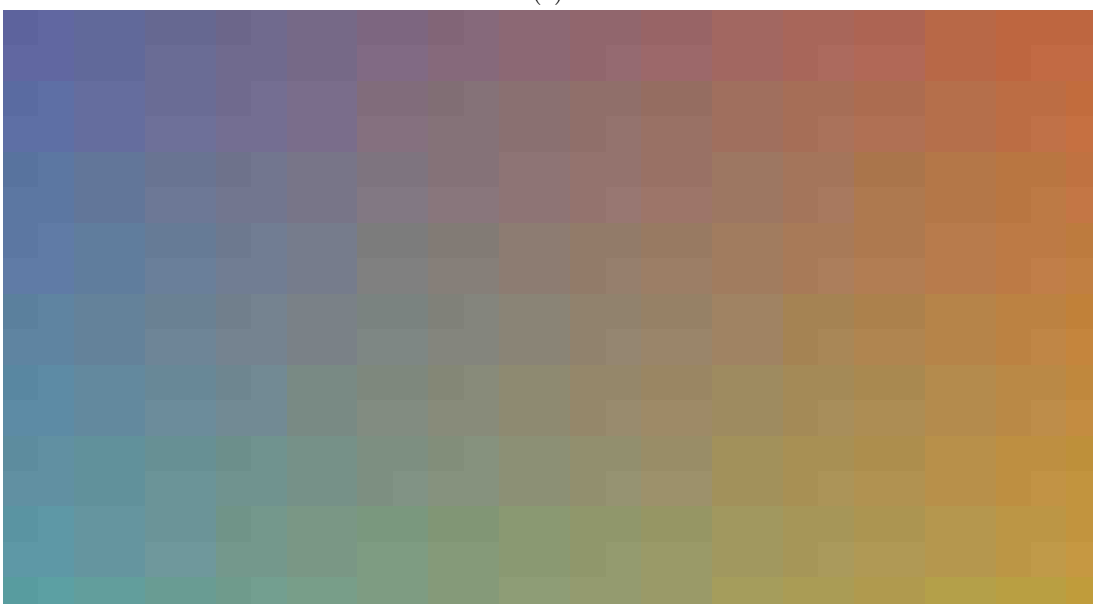


(f) Subject 6

Figure 19: Quality choices distribution per subject

B.2 Image Samples

(a)



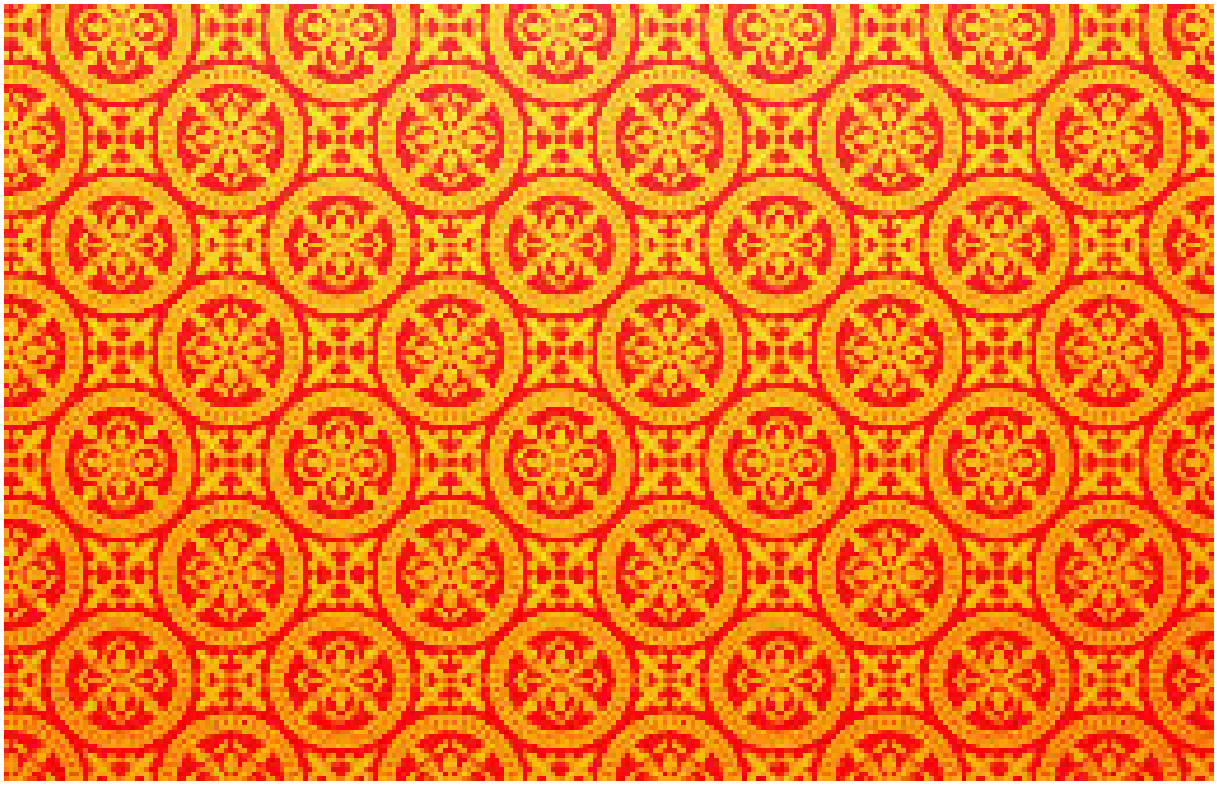
(b)

Figure 20: (a) Original Image (b) Compressed Image with blocking artifacts

(a)

(b)

Figure 21: (a) Original Image (b) Compressed Image with ringing artifacts



(a)

(b)

Figure 22: (a) Original Image (b) Compressed Image with overall quality degradation

C Hypothesis Validation Experiment Results

The results show the quality level choices of each of the 6 subjects for the presented set of images.

Image	Subject1	Subject 2	Subject 3	Subject 4	Subject 5	Subject 6
1-val.png	90	85	80	85	85	85
2-val.png	90	95	70	65	80	80
3-val.jpg	85	90	75	85	75	85
4-val.jpg	70	85	65	60	60	70
5-val.png	75	85	80	65	75	85